Exploring Heterogeneous Computing with Advanced Path Tracing Algorithms

André Oliveira, César Perdigão, Luís Paulo Santos, Alberto Proença
Dept. of Informatics – Algoritmi Research Center, Universidade do Minho, Portugal
Email: psantos@di.uminho.pt

***Abstract*—The CG research community has a renewed interest on rendering algorithms based on path space integration, mainly due to new approaches to discover, generate and exploit relevant light paths while keeping the numerical integrator unbiased or, at the very least, consistent. Simultaneously, the current trend towards massive parallelism and heterogeneous environments, based on a mix of conventional computing units with accelerators, is playing a major role both in HPC and embedded platforms. To efficiently use the available resources in these and future systems, algorithms and software packages are being revisited and re-evaluated to assess their adequateness to these environments.**

This paper assesses the performance and scalability of three different path based algorithms running on homogeneous servers (dual multicore Xeons) and heterogeneous systems (those multicore plus manycore Xeon and NVidia Kepler GPU devices). These algorithms include path tracing (PT), its bidirectional counterpart (BPT) and the more recent Vertex Connect and Merge (VCM). Experimental results with two conventional scenes (one mainly diffuse, the other exhibiting specular-diffuse-specular paths) show that all algorithms scale well across the different platforms, the actual scalability depending on whether shared data structures are accessed or not (PT vs. BPT vs. VCM).

***Keywords* —** global illumination, path space based integration, heterogeneous parallel computing

I. Introduction

Heterogeneous Parallel Computing is an emerging trend in today's computing solutions. The demand for constantly increasing computational power has been met by manufacturers with a range of highly parallel computing devices, including multicore/multisocket CPU architectures, and manycore solutions such as Xeon Intel Phi and Graphics Processing Units (GPU), usually packaged as co-processing expansion boards. More specialized architectures, such as DSPs and FPGAs, are also widely available. Leveraging the whole heterogeneous set of computational resources available on a single machine in order to reduce parallel applications' execution times is thus an obvious course of action.

However, heterogeneous systems pose a number of challenges, which make it difficult for programmers to exploit the available computing power. Typically accelerators have disjoint address spaces among themselves and the host CPU, usually interconnected with a limited bandwidth bus which is a potential performance bottleneck. Furthermore, different architectures usually exhibit different execution and programming models and are deployed with different programming languages and development tools, severely impacting on both code and performance portability. Additionally, the application's workload has to be distributed and balanced among the multiple devices, and, within each device, among its multiple computing units; this leads to multilevel scheduling, which must be effectively handled in order to achieve acceptable performance levels [1]. To efficiently use the available resources in these and future systems, algorithms and software packages have to be revisited and re-evaluated to assess their adequateness to these environments.

Simultaneously, graphics applications keep increasing overall quality demands. These demands include realism (implying more accurate simulation of light transport), image resolution, scene complexity and multi-views for enhanced depth perception, all impacting on the computing power required to timely render such images. Efficient exploitation of the computational resources available on current heterogeneous systems is mandatory in order to meet the quality and timing requirements of modern graphics applications. Well known algorithms have thus to be re-thought and their performance on such heterogeneous systems thoroughly assessed.

Within the physically-based global illumination research community, path space integration based rendering algorithms [2], [3] have witnessed a renewed interest. These algorithms are unbiased, converging to the correct solution as the number of samples (traced paths) increases and tends to infinity. Although the range of efficiently simulated light transport phenomena has improved dramatically compared to the original algorithms, mainly due to bidirectional information propagation and multiple importance sampling strategies, some transport phenomena still exhibit a very low probability of being captured and the convergence rate might be slow for applications with demanding timing constraints. The renewed interest on these approaches results from their combination with biased, but consistent, algorithms, such as progressive photon mapping [4], [5]. The "Vertex Connect and Merge" (VCM) algorithm [6] is an example of such symbiosis between path integration and density estimation algorithms, allowing for better convergence rates and simulation of such low probability directional transport phenomena (such as specular–diffuse–specular), while maintaining guarantees of convergence to the correct solution as more samples are taken. Performance results for such approaches are, however, known only for homogeneous systems or, at most, hybrid approaches, where the CPU controls a single GPU and performs some

978-1-5090-5387-2/16/$31.00

Fig. 1. Sponza reference image

specialized tasks, which do not map onto the GPU's Single Instruction Multiple Threads (SIMT) computation model [7].

This paper addresses the development and execution of three major path space integration rendering algorithms on highly parallel heterogeneous servers. The algorithms include path tracing (PT) [2], bidirectional path tracing (BPT) [3], [8] and VCM [6]. The parallel heterogeneous servers include dual multicore Intel systems coupled to either manycore Xeon Phi Knights Corner or NVidia Kepler GPU accelerator boards.

The contributions of this paper include:

- the proposal of a two level workload decomposition for the above cited algorithms, which allows for the exploitation of multiple heterogeneous parallel devices while maintaining memory requirements within satisfiable bounds;
- the assessment of the proposed decomposition's performance and scalability on two highly parallel heterogeneous servers.

This paper is organized as follows: section II describes the rendering algorithms. The next section describes in detail the diversity of devices used on the heterogeneous servers, followed by a section on the proposed workload decomposition. Results are then presented and analyzed. The paper finishes with some concluding remarks.

II. Path Tracing Algorithms

Kajiya's formulation of the rendering equation as a Fredholm integral of the second kind [2] allowed for the application of a range of mathematical tools to numerically approximate the solution of this equation. In the same seminal paper the author identifies radiosity as a Finite Element Method, distributed ray tracing [9] as a Monte Carlo integration technique and proposes the path tracing algorithm based on a Neumann series expansion of the rendering equation. This algorithm would be the first of a series of path space–based integration algorithms proposed over the years. On this paper three such algorithms are analysed: path tracing (PT), bidirectional path tracing (BPT) and vertex connection and merge (VCM). The next subsections give a brief description of each of these algorithms; the reader is referred to the vast bibliography on the subject [10].

A. Path Tracing

Path tracing (PT) consists in, for each sample on the image plane, performing a random walk from the camera, through the image plane, into the scene. The path starts with the primary ray and, at each intersection point, the next ray's direction is selected according to some probability density function. On its unbiased form the random walk terminates, i.e., the path finishes, using a Russian Roulette approach. Among the many optimizations, the most common is next event estimation, i.e., at each intersection point evaluate direct lighting using a number of shadow rays, therefore building in fact a tree of paths rather than a single path. Path tracing progresses from the camera towards the light sources, thus gathering radiance for a single point on the image plane – these are often referred to as camera paths.

Fig. 2. Living Room reference image

B. Bidirectional Path Tracing

Bidirectional Path Tracing [3], [8] extends path tracing by tracing a light path for each camera path. The light path originates at the light source and performs a random walk into the scene thus propagating radiance. By properly combining each light path vertex with each camera path vertex a large number of different paths is simulated with minimal computational effort. Veach [3] also proposes that the weighting of each of these paths must take into account all the different ways upon which these could have been generated, thus introducing Multiple Importance Sampling. Propagating radiance from the light sources allows for the efficient simulation of specular to diffuse light transport phenomena, i.e., caustics. Note that the BPT algorithm is initiated for a given sample (point) in the image plane; this is what triggers the two random walks. But when the light path vertices are connected to the camera, any point on the image plane can be intersected. Therefore, multiple trees of paths are generated, contributing to different pixels which can not be determined beforehand.

C. Vertex Connection and Merge

Vertex Connection and Merge (VCM), introduced in 2012 by Georgiev et al. [6], aims at efficiently simulating specular–diffuse–specular light transport phenomena (i.e., reflected caustics), which have probability zero of being traced with perfectly specular BRDFs and unbiased methods. The authors combine progressive photon mapping (PPM) [4], [5] with bidirectional path tracing to develop a consistent numerical integrator to the rendering equation. The photon mapping approach, based on density estimation, is recast as a path integration approach, therefore allowing the combination of these two algorithms. Being based on PPM, the VCM algorithm is iterative. Each iteration consists generating a photon map with a constant number of photons followed by a BPT pass. The photons' radius of validity decreases as the iteration index increases, thus rendering the method consistent (it will converge to the correct solution as the number of total photons approaches infinity and the radius of validity approaches zero). Note that VCM iterations are completely independent from each others, since the photons validity radius only depends on the iteration index; iterations can thus be computed on any order.

III. Heterogeneous Servers

Two heterogeneous servers were used to obtain the results reported in section V, each including two 12-core Intel Xeon coupled to either manycore Intel Xeon Phi Knights Corner or NVidia Kepler GPU devices.

The multicore Intel Xeon E5-2695v2 [11] packages 12 CPU cores, running at 2.4 GHz, with support for simultaneous execution of up to 24 threads due to HyperThreading. Each core includes 2x32 KB L1 separate caches for data and instructions, and 256 KB unified L2 cache. The 30 MB's unified L3 cache is shared among the 12 cores. Each of the used servers has two 12-core Intel Xeon E5-2695v2 CPUs with 64 GBs central memory; with Hyperthreading this amounts to 48 logical cores on each server.

The Intel Xeon Phi 7120 Knights Corner [12] is a co-processor packaging 60 cores running at a peak 1.33 GHz,

with 16 GB of memory and a maximum of 352 GB/s memory bandwidth. Each core includes a vector processing unit (VPU), featuring a 512-bit SIMD instruction set, capable of executing 16 single-precision (SP) or 8 double-precision (DP) operations per cycle. Fused Multiply-Add instructions are also executed on a single cycle, increasing the FLOP count to 32 SP or 16 DP per clock cycle. Each of these cores supports 4-way simultaneous multithreading, for an expected performance peak at 240 threads (all other factors being equal). The first heterogeneous server has two Xeon Phi 7120 Knights Corner accelerator boards.

The global illumination engine developed to support the path tracing algorithms on both Intel platforms uses Intel Embree Ray Tracing Kernels [13]. The kernels are optimized for global illumination algorithms, selecting at runtime the acceleration structure build and traversal more appropriate for the underlying Intel CPU or co-processor. An important feature of Embree is that it leverages the different vector instruction set extensions supported by Intel processors, while simultaneously containing algorithms to handle incoherent workloads, such as those resulting from Monte Carlo path tracing approaches. Within each Intel multi/many core device parallelism is managed using Intel Thread Building Blocks [14], a C++ template library for task parallelism.

NVIDIA's Tesla K20m accelerators are PCI boards including the GK110 graphics processing unit (GPU) and 5 GB of GDDR5 SDRAM. The GK110 GPU on these boards is packaged with 13 Streaming Multiprocessors (SMX), each with 192 CUDA cores, for a total of 2496 such cores, running at 706 MHz [15]. Each SMX includes 64 KB configurable shared memory and L1 cache and an additional 48 KB of read only data cache. The 1536KB L2 cache is shared among all SMXs. The path tracing algorithms running on the NVIDIA devices run on top of the OpTiX ray tracing engine [16], [17]. The second used server is endowed with two such boards.

IV. Workload Decomposition and Scheduling

The workload associated with each of the rendering algorithms is decomposed at two different levels. At the upper level the workload is decomposed into iterations, at the lower level the decomposition depends on the device where the corresponding iteration is scheduled.

For the three algorithms an iteration implies evaluating all paths spawned by shooting one primary ray per pixel. The end result of an iteration is effectively a low quality image, with a sampling rate of 1 sample per pixel. Each device maintains its own local frame buffer, where the results of different iterations are accumulated. Iterations are assigned to devices on a demand driven basis and each device is operating on a single iteration at any instant of time. There is no contention among different iterations accessing the device local frame buffer, since these are not processed simultaneously. The final frame buffer results from the integration of the devices' local buffers; this operation is performed in the host CPU after all iterations have finished, to avoid communication overheads with the devices - this is a sequential operation (given the reduced number of devices), which does not require any memory access control mechanism.

All the physical/logical cores of multi-socket central processing units are treated as a single device and assigned the same iteration. Also all the physical/logical cores of each Xeon Phi device are assigned a single iteration. This is due to the fact that the VCM algorithm photon map requires approximately 0.5 GBs of memory (1024*768 light paths with an average of 10 photons per light path), hindering the possibility of assigning an iteration (and consequently a different photon map) to each of these devices' cores. Each GPU accelerator present on the heterogeneous system is also assigned an iteration following the same demand driven policy. Besides the above explained memory limitations, the ray tracing engine used within the GPUs is OpTiX [17], inhibiting the simultaneous execution of more than one iteration per graphics board. At this upper level of workload decomposition and scheduling, tasks – iterations – are strongly related to memory address spaces: computational resources sharing the same address space (cores within the central processing unit, cores within the Xeon Phi, elementary processors within the GPU) are assigned a single iteration, such that memory consumption, i.e., the number of photon maps, does not increase with the number of such resources per device (we have in fact one photon map per device).

The lower, secondary, level of scheduling depends on the particular device. NVIDIA's OpTiX, used in the GPUs, is responsible for managing the workload within that device and performs it at the ray level, beyond the programmer's control. For the remaining devices workload decomposition is achieved by partitioning the image plane into a large number of tiles and assigning them to the processing cores on a demand driven basis. For the VCM algorithm this is preceded by the photon shooting stage, with the number of light paths uniformly distributed among the cores.

Within these devices, different tasks, and consequently the various threads, share different data structures depending on the rendering algorithm. On the PT case, the tree of paths spawned by one primary ray contributes only to the corresponding pixel. Threads share no writable data structure, since each task writes only to the pixels of the frame buffer corresponding to the image space tile assigned to it. BPT and VCM tasks can write anywhere in the device frame buffer, since the connection of the light path vertices to the camera results on potential contributions to any pixel in the image plane. Concurrent access control among the cores of the Intel devices is performed by using a single spin mutex, which protects the whole image plane – with Intel's Thread Building Blocks using a single mutex proved to be as efficient, performance-wise, as using multiple mutexes to protect different regions of the image plane. In GPUs atomic operations are used. VCM also requires building a photon map per iteration. On the photon tracing stage, each thread is responsible for a subset of the light paths. An hash grid is used to speedup range searches during the rendering stage, therefore upon creation of a new photon the respective voxel photon counter is atomically

incremented. When all photons have been created voxels are linked to photons using an offset vector and atomic operations whenever required.

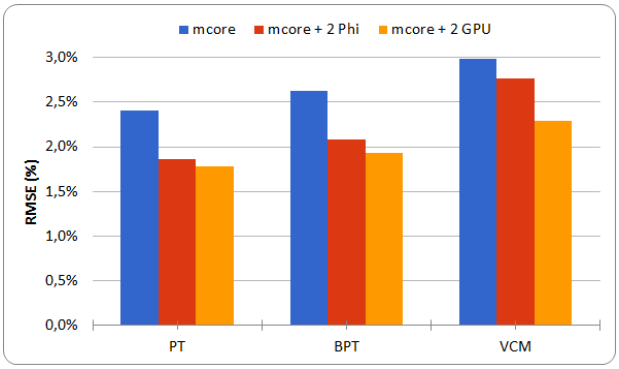
V. Performance Analysis

A. Methodology

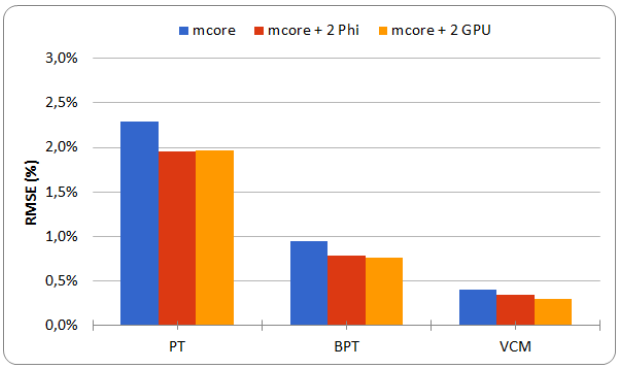
Experimental results have been obtained by executing the three path space integration algorithms (PT, BPT and VCM) with two different scenes on two heterogeneous servers.

Each experiment consists on rendering one scene with a given algorithm on a given server. The rendering time is always five (5) minutes and the number of completed iterations is registered. The performance metric is thus number of iterations per five minutes. The number of iterations correlates with image quality, since it is well known that variance reduces with the number of total samples, since all algorithms are unbiased or consistent. The reported values are obtained using the K-best methodology out of a maximum of 10 executions, with a tolerance of 3%.

The well-known "Sponza" and "Living Room" scenes were selected to generate the results because they represent two different challenges with respect to the light transport phenomena involved. "Sponza" is mainly diffuse and lighten by daylight, whereas the "Living Room" is an interior scene, with specular reflective and transmissive materials, which result on L*SDS*E light paths (specular-diffuse-specular, i.e., reflected caustics), well captured by the VCM algorithm. All images were rendered at a resolution of 1024x768 pixels. The reference images used to validate the correctness of the algorithms' implementations and to verify convergence with the number of iterations were obtained by running the VCM algorithm during 4 hours on a dual 12-core Intel Xeon E5-2695 server – see figures 1 and 2. Figure 3 presents the root mean square error (RMSE) obtained with each algorithm for each scene and different hardware configurations; the RMSE is evaluated for 5 minutes executions relatively to the reference images. The values are less or equal to 3% in all cases, confirming convergence.



(a) Sponza



(b) Living Room

Fig. 3. Root Mean Square Error w.r.t. the reference images

B. Results Analysis

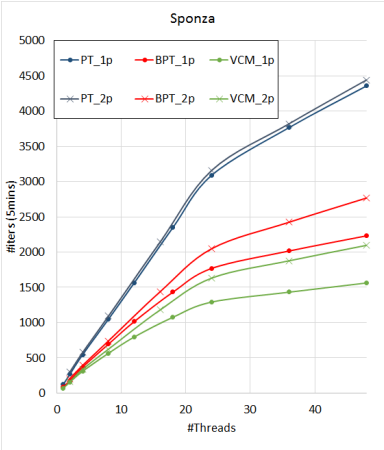
1) Scalability on multicore devices: Figure 4 depicts the number of iterations completed within 5 minutes of execution time for different numbers of threads on the two 12-core Intel Xeon E5-2695 devices. Results are presented for the two scenes and for two different configurations. The data series labelled as "1p" were obtained by using a single process as a container of all threads; the data series labelled as "2p" were obtained by using two processes each associated with half the threads. Whereas in the former threads are allowed to migrate between cores on different devices (or sockets), in the latter this migration is not allowed.

All three algorithms scale well, although it is noticeable that PT scales far better that the other two. PT is embarrassingly parallel in the sense that no written data structure is ever shared among threads. BPT implies sharing the frame buffer, since a light path can write onto any pixel; VCM implies sharing the photon map for both writing and reading. The slope of the performance curve of each algorithm is therefore dependent on the level of shared data structures. It is also noticeable that no inflection point has been reached, in the sense that performance keeps increasing with the number of threads - it would be interesting to analyse when and which algorithms would reach this inflection point first, if more cores were available. The bending of the curve when the number of threads increases from 24 to 25 is evident. This is the point where HyperThreading starts being used; the sharing of resources on this form of simultaneous multithreading does not allow for a doubling in performance.

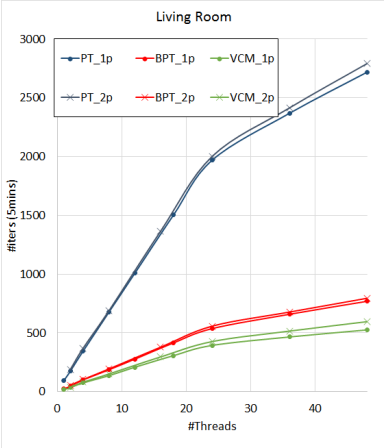
The 2 processes configuration exhibits a significant impact on performance relatively to the 1 process case. Since threads are not allowed to migrate between sockets, data reutilization on the caches shared among cores within the same device is larger. Locality of data acesses is therefore a relevant issue to take into consideration.

2) Scalability within the manycore device: Figure 5 depicts the number of iterations completed within 5 minutes of execution time for different numbers of threads on the Intel Xeon Phi 7120 device. This is a 60 physical cores device, each core capable of up to 4-way Hyperthreading (HT). Results are thus depicted for 1, 15, 30 and 60 threads (no HT), 120 threads (HT 2-way), 180 threads (HT 3-way) and 240 threads (HT 4-way).

Path tracing scales far better than the other two algorithms.

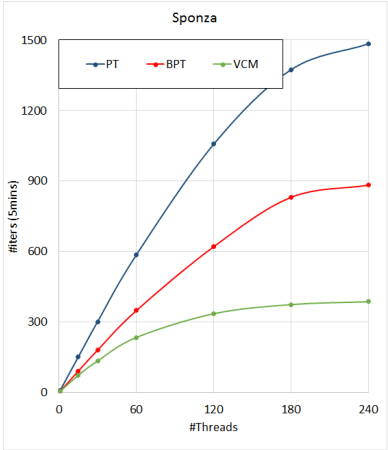


(a) Sponza

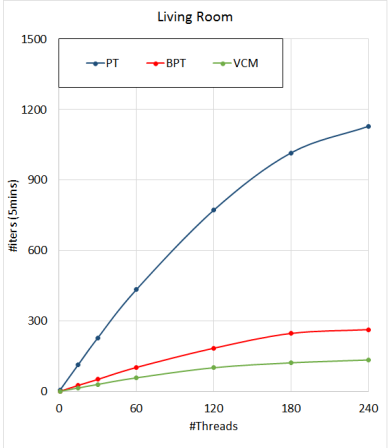


(b) Living Room

Fig. 4. Scalability on Intel Xeon muticore devices



(a) Sponza



(b) Living Room

Fig. 5. Scalability within the Intel Xeon Phi manycore device

Although none of the algorithms presents an inflection point up to 240 threads (i.e., performance does not decrease with the number of threads), it is clear that maximum performance has been achieved for both BPT and VCM. For instance, in figure 5b VCM's total number of iterations is roughly the same for 180 and 240 threads. Adding computational resources no longer results on significantly increased performance due to data sharing.

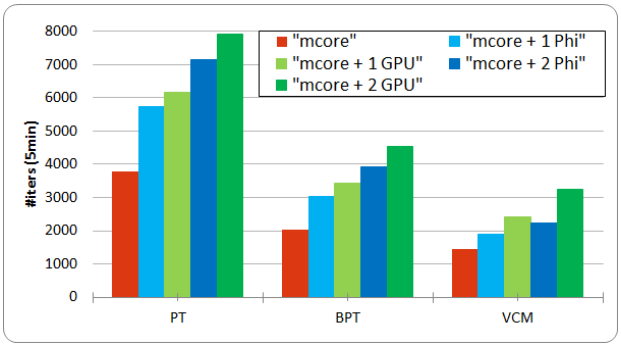
3) Heterogeneous Scalability: Figure 6 depicts the number of iterations completed within 5 minutes of execution time for different combinations of accelerator devices.

The dual multicore Xeon E5 is, for all reported scenes and algorithms, the device that contributes with a larger fraction of iterations than any other single device. GPUs clearly outperform the intel Xeon Phi devices (except for one single case). The most important conclusion which can be drawn from this set of results is that, with up to two additional GPUs or Xeon Phi, performance increases for all reported path space integration algorithms.

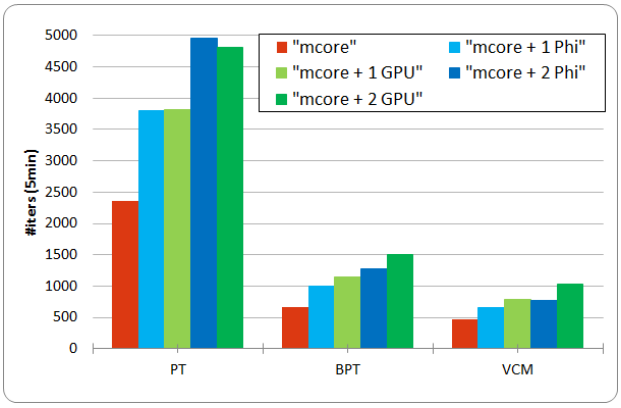
In order to better illustrate how this performance gain behaves figure 7 presents speedup over the multicore configuration, i.e., for each scene and algorithm, the ratio between the number of iterations obtained including the multicore CPUs plus the accelerators and the number of iterations using the multicore alone. Two additional results become now more evident. First, for each algorithm and for each kind of device speedup always increases with the number of such devices; this raises the question of up to which number of devices will speedup maintain this increasing trend. Second, all three algorithms scale with the added heterogeneous computing power. Whereas figure 6 could lead to the conclusion that PT scales much better than the other two, this is not true. The speedups obtained with GPUs are impressive for all algorithms, and, in fact, PT seems to be slightly less scalable than the alternatives – a more thorough study of this phenomenon is required, to understand whether this results from these particular data sets and/or servers and whether this trend would be maintained with a larger number of accelerator boards.

VI. Conclusion

This paper addresses the efficient execution of three path space integration based rendering algorithms (path tracing, bidirectional path tracing and vertex connect and merge) on highly parallel heterogeneous servers. The heterogeneous

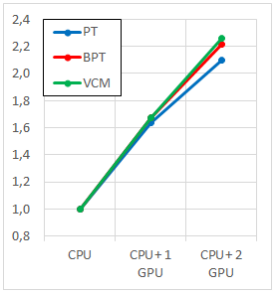


(a) Sponza

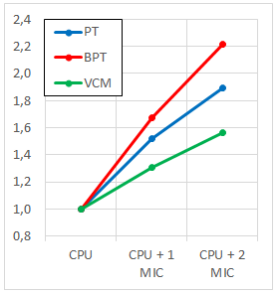


(b) Living Room

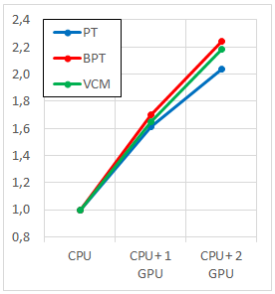
Fig. 6. Scalability with multiple heterogeneous devices



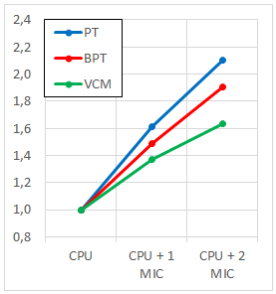
(a) Sponza + GPU



(b) Sponza + Phi



(c) Living Room + GPU



(d) Living Room + Phi

Fig. 7. Speedup with multiple heterogeneous devices

servers include dual multicore Intel Xeon with either up to two manycore Intel Xeon Phi Knights Corner or NVidia Kepler GPU accelerator devices.

The huge memory requirements of the photon maps associated with VCM, which is based on progressive photon mapping, prevent the scheduling of an iteration to each core of the parallel devices. A two level workload decomposition is therefore proposed, which entails assigning a single iteration to a device (on a demand driven basis) and then partitioning the image plane among that device's parallel computing resources – again on a demand driven basis. The exception are the GPUs, where OpTiX is responsible for all workload decomposition and distribution within each GPU – this is performed at the level of ray packets, to the best of our knowledge. It is shown that this workload decomposition approach, allows for the utilization of highly parallel devices (48 threads for the Intel Xeons, 240 threads for the Intel Xeon Phi), while maintaining memory requirements within satisfiable bounds.

Scalability results show that all three algorithms scale well and no inflection point is reached for the number of used devices, i.e., performance increases with additional computational resources. However, scalability is clearly related to the degree of shared data among threads, with PT exhibiting the best results. Results also show that assuring locality of data accesses impacts significantly on overall performance. Using two processes, instead of one, to avoid inter-socket threads' migration on the dual multicore Xeon (i.e., assuring thread affinity) increases performance, by better exploiting each Xeon caches.

It is clearly demonstrated that these heterogeneous servers have an interesting potential for the efficient execution of path tracing based algorithms. Even though scalability for the more sophisticated algorithms – requiring shared data structures – seems to be approaching the limit – technology has already moved on and it would be interesting to exploit a larger number of accelerators (>2) and more heterogeneous servers (multicore CPUs plus manycore Xeon Phi plus GPUs). The new Intel Xeon Phi device, the Knights Landing, displays interesting features worth to exploring. The more eye catching are the 64-bit Atom cores replacing the outdated P54, the dual AVX-512 units per core, a larger number of cores (72) organized as 36 dual-core tiles interconnected through a 2D mesh, and the 16GB on-package memory that can be configured as cache or RAM. Additionally, the possibility of running code without an host CPU provides an excellent opportunity to soon enrich this comparative evaluation of the same three algorithms scalability on this new device.

Acknowledgments

This work was supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT (Fundação para a Ciência e Tecnologia) within Project Scope (UID/CEC/00319/2013), by the Cooperation Program with the University of Texas at Austin and co-funded by the North Portugal Regional Operational Programme (ON.2 - O Novo Norte), under the National Strategic Reference Framework, through the European Regional Development Fund.

We would like to thank Jaroslav Křivánek for making the "Living Room" model available.

REFERENCES

[1] R. Ribeiro, J. Barbosa, and L. P. Santos, "A framework for efficient execution of data parallel irregular applications on heterogeneous systems," *Parallel Processing Letters*, vol. 25, 2015. [Online]. Available: http://dx.doi.org/10.1142/S0129626415500048

[2] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/15922.15902

[3] E. Veach, "Robust monte carlo methods for light transport simulation," Ph.D. dissertation, Stanford University, 1998.

[4] T. Hachisuka, S. Ogaki, and H. W. Jensen, "Progressive photon mapping," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 130:1–130:8, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1409060.1409083

[5] C. Knaus and M. Zwicker, "Progressive photon mapping: A probabilistic approach," *ACM Trans. Graph.*, vol. 30, no. 3, pp. 25:1–25:13, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1966394.1966404

[6] I. Georgiev, J. Křivánek, T. Davidovič, and P. Slusallek, "Light transport simulation with vertex connection and merging," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 192:1–192:10, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2366145.2366211

[7] T. Hachisuka and H. W. Jensen, "Parallel progressive photon mapping on gpus," in *ACM SIGGRAPH ASIA 2010 Sketches*, 2010, pp. 54:1–54:1. [Online]. Available: http://doi.acm.org/10.1145/1899950.1900004

[8] E. P. Lafortune and Y. D. Willems, "Bi-directional path tracing," in *Proceedings of Compugraphics '93*, Alvor, Portugal, 1993, pp. 145–153.

[9] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '84. New York, NY, USA: ACM, 1984, pp. 137–145. [Online]. Available: http://doi.acm.org/10.1145/800031.808590

[10] M. Pharr and G. Humphreys, *Physically Based Rendering: from Theory to Implementation*, 2nd ed. Morgan Kaufmann, 2010.

[11] "Intel® xeon® processor e5-2695 v2," accessed: 2016-10-01. [Online]. Available: http://ark.intel.com/products/series/75291/Intel-Xeon-Processor-E5-2600-v2-Product-Family

[12] "Intel® xeon® phi™ coprocessor 7120a," accessed: 2016-10-01. [Online]. Available: http://ark.intel.com/products/series/75809/Intel-Xeon-Phi-Coprocessor-7100-Series

[13] "Embree: High performance ray tracing kernels," accessed: 2016-10-01. [Online]. Available: http://embree.github.io/

[14] "Intel thread building blocks," accessed: 2016-10-01. [Online]. Available: https://www.threadingbuildingblocks.org/

[15] NVIDIA, "Nvidia's next generation cuda compute architecture: Kepler tm gk110," White Paper. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[16] "Nvidia's optix ray tracing engine," accessed: 2016-10-01. [Online]. Available: https://developer.nvidia.com/optix

[17] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: A general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 66:1–66:13, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778803